122nd ASEE Annual Conference & Exposition
June 14 - 17, 2015
Seattle, WA
Paper ID #13644

Project-Based Learning with Single-Board Computers

Dr. Joseph Daly Steinmeyer, Massachusetts Institute of Technology

Joseph Steinmeyer is currently a lecturer in the EECS Department at MIT. He obtained his BS in EECS from the University of Michigan in 2008, and his MS and PhD (also in EECS) from MIT in 2010 and 2014, respectively. His research interests currently center around neuroscience and engineering, educational technology development, and STEM curriculum development at both the high school and college level.

Project-Based Learning with Single-Board Computers

Introduction

Project-based learning (PBL) has been shown to be effective in the STEM fields[1,2]. In implementing PBL of Electrical Engineering and Computer Science (EECS) topics for middle and high school-level enrichment programs, significant thought needs to go into determining which sub-topics in the EECS field should be covered in a curriculum in order to enable students to undertake projects of sufficient and satisfying complexity. One solution is to focus exclusively on either the software (programming) or hardware (sensors, circuits, etc.) side of EECS, having what amounts to either a computer science curriculum or a circuit theory curriculum. In the last few years, however, we've been focusing on in integrating both hardware and software engineering into EECS PBL at the secondary level. We have been carrying out this work with classes of rising high school seniors in a number of summer enrichment programs run through the Office of Engineering Outreach Programs (OEOP) at the Massachusetts Institute of Technology. In the last two years we have particularly been exploring ways of deploying single-board computer platforms including the BeagleBone Black and Raspberry Pi as means of providing a flexible, enriching, and open-ended project-based learning experiences that provides significant exposure to both hardware and software development. In this paper we will discuss the coursework infrastructure we developed along these lines for three separate summer STEM enrichment programs: a six week program, a one week program, and a three hour workshop. We follow this with some preliminary student feedback, plans for expansion, and plans for quantitatively assessing the efficacy of the curriculum in the long term.

The OEOP at MIT has run its Minority Introduction to Technology, Engineering, and Science (MITES) program for the past forty years, serving rising seniors from across the country coming from traditionally underserved and underrepresented backgrounds. MITES is a six-week residential program where students take a number of courses including calculus, physics, life sciences, humanities, and a project course. In the last five years, the OEOP has expanded its outreach efforts through the creation of several additional programs, including the Engineering Experience at MIT (E2@MIT), a one-week residential program heavily focused on only a project course, as well as the MIT Online Science, Technology, and Engineering Community (MOSTEC), a six-month online curriculum with a one-week residential portion that includes STEM courses, seminars, and other activities. Within each of these programs, we were tasked with creating EECS-themed PBL curricula (a project course for MITES, the primary course for E2@MIT, and a series of workshop activities for the residential portion of MOSTEC). In addition we have created an online EECS MOSTEC curriculum with separate themes that we detail elsewhere[3].

Curriculum Design and Considerations

Starting with only the MITES project course in 2008, the EECS curriculum was initially solely hardware-focused, with students learning basic circuit theory and creating projects

composed of discrete components (resistors, capacitors, basic integrated circuits, etc.) implemented on breadboards. Based on student feedback from these first few years, and with the advent of two additional programs (the E2@MIT program and the MOSTEC workshops), we began to integrate software elements, first utilizing the popular Arduino platform in 2010 and then merging with single-board computers in 2013[4–6]. As we've added in this software engineering component, we've placed a priority in maintaining a significant hardware aspect to all student work. This has necessitated streamlining of the platforms and environments worked in by students to ensure a sufficient levels of engagement. We've used several self-imposed guidelines as we developed the curriculum:

- We've found an increasing number of our students come into our programs with some prior exposure to robotics via either FIRST Robotics or other they have been involved in[7–12]. We have therefore avoided using robotics as a vehicle of STEM exploration and teaching. This ensures course material is new to all students and it levels the playing field for the class so students without robotics exposure aren't at a disadvantage. By keeping the vehicle of EECS exploration more generalized, it also allows students to pursue topics of interest to them, and hopefully in the process ensure greater motivation for a broader range of students. Music-based labs and final projects, for example, have been a popular class of projects that students have pursued in the past.
- When working with single-board computers and Arduino (in the past), we avoided the use of add-on pre-made boards (shields and capes) in order to maximize the chance for component-level circuit analysis and design, implementation, and experimentation. Breadboarding is a key component in the curricula we develop, and final projects must involve some form of discrete circuit component.
- We have emphasized the concepts of sensors, signals, and signal processing when teaching programming and electronics. Many labs, homework exercises, and activities involve interpreting signals generated by using sensors and circuits of the student's creation, interpreting those signals using programming, and making design decisions based off of those interpretations.
- All laboratory exercises in the first half of the courses are designed to be carried out in groups of two or three, and be significantly open-ended (see Tables 1, 2, and 3). This prepares students for the open-ended final design project in the second half of the course. In the MITES Electronics curriculum (long residential program), student final projects take place over about two-and-a-half weeks and are extremely open-ended, with students building projects of their own design with few end-goals specified (other than it be "something they've always wanted to build" and of sufficient complexity). In the shorter one-week E2@MIT curriculum, loose project assignments are provided for them, however students invariably customize and bend specifications throughout the design and implementation process. In the three-hour MOSTEC workshops, projects are similar to a one-day laboratory that would be found in the longer MITES and E2 programs, but are still left open-ended.

Motivation for Single-Board Computers

We used the Arduino environment for several years in MITES and E2@MIT as well as to a lesser degree in our first iterations of MOSTEC workshops[5]. Excellent overviews of Arduinos exist in the literature[13,14]. In using Arduinos, we encountered several limitations in the context of the courses we ran:

- Arduino programming relied upon external or student-provided computers for programming. The courses we ran had to take place either in rooms with available computers or require students to bring their own laptops. In either case time had to be spent to prep the computers for running the Arduino Integrated Development Environment (IDE), which in the case of student-owned laptops would invariably come out of class time.
- Arduino programming had to be carried out in the Wiring programming language/environment. In other courses both at and affiliated with MIT, we have been teaching the Python programming language[15,16], and we desired to transfer/borrow this infrastructure and coursework into the STEM curricula developed for these EECS courses.

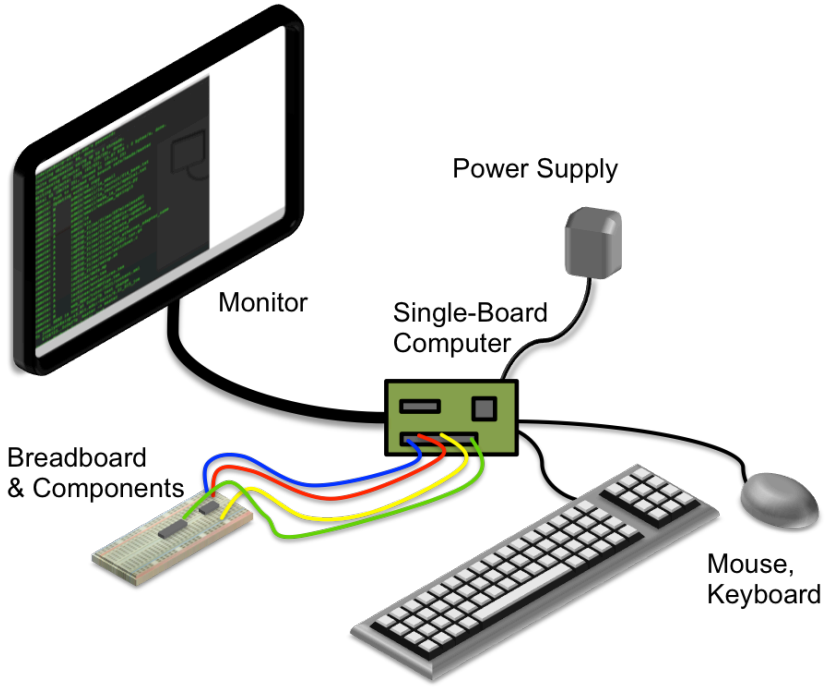


Figure 1. Single-Board Computer Prototyping Platform.

We investigated using single-board computers new to the market in early 2013 including the Raspberry Pi[6], and the BeagleBone Black single-board computers as a solution to the problems above with Arduinos. Samples of both computers were purchased, and we developed experimental platforms similar to that shown in Figure 1 with a monitor, keyboard, and mouse being directly hooked up to the computer. The single-board computers could run their own operating system, with the intention that students would interface with the computers input/output connections via hookup wires that would go to

their hardware components implemented on a breadboard. The result was a completely stand-alone development platform to be used by students that was uniform for the entire class. This also allowed the courses to be taught more easily in multiple classroom settings since the entire development environment was self-contained, requiring only power outlets and Internet access.

In 2013, there were two major choices for single-board computers at very similar price points: the Raspberry Pi and the BeagleBone Black[17]. From initial testing, the BeagleBone Black appeared the better choice, due to the larger number of input and output pins, and on-board integrated compatibility with analog and digital signals. In-house benchmark tests run by us also showed the BeagleBone Black to run more quickly than the Raspberry Pi when used in the self-contained prototyping platform in Figure 1. Additionally at the time, Raspberry Pis were in short supply. We therefore settled on the BeagleBone Black as the single-board computer of choice.

Curriculum Design (MITES and E2@MIT)

While the MITES and E2@MIT electronics programs differed significantly in overall duration, with the former being five weeks and the latter being one week, they both followed a similar curriculum path shown in Table 1 below. Because students in the E2@MIT program focus almost exclusively on their project course and MITES students spend roughly one-fifth of their time on the project course, the total amount of hours devoted to the EECS course is very similar between the two programs. In 2014, total student course time was tallied at approximately 40 hours for the E2@MIT program, while for MITES the total time throughout the course was closer to 55 hours. Because of these similarities, both courses cover similar amounts of material, however the number of guided laboratory exercises was fewer in E2@MIT than in either iteration of MITES (Table 3 vs. Table 4). In both courses, we chose to teach elements of software engineering simultaneously with hardware engineering to immediately enable lab exercises involving both types of activities.

For both MITES and E2@MIT, the first half of the course was built around lab exercises devoted to developing and exploring hardware and software concepts techniques. During this half of the course, students participated in an instructor-led discussion to introduce concepts that were going to be encountered in lab that day. This portion of the class was about twenty minutes for MITES and thirty minutes for E2@MIT. Following this, students spent the remainder of the class (ninety minutes in MITES and three hours in E2@MIT) carrying out hardware and software laboratory exercises with the single-board computer platforms covering topics shown in Tables 2 and 3. Each lab was designed to be about two-thirds guided with an open-ended final portion where students could pursue a number of suggested projects expanding on the themes for the day or come up with their own additional projects and modifications. An extract from one lab handout is shown in Figures 2 and 3. Students were provided nightly homework sets that were carried out in an online tutor environment (see discussion on Python below). Homework questions focused more on filling in gaps of knowledge in programming, circuits, and signals in preparation for lab work the following day.

MITES	E2@MIT	Topics
Week 1	Day 1	Programming, Inputs, Outputs, Control
Week 2	Day 2	Programming Signals, Amplification
Week 3	Day 3	Feedback Control/Final Project
Week 4	Day 4	Final Project
Week 5	Day 5	Final Project
Final Day	Final Day	Final Presentations

Table 1. Overview of similarity between MITES and abbreviated E2@MIT programs

Lab Number	MITES 2013	MITES 2014
1	Linux, Python I, Digital Outputs	Linux, Python, Digital Outputs
2	Python II, Digital I/O	Python II, Digital Inputs, Outputs
3	Python III, Signals	Audio Amplification with Pygame
4	Motor Control, Amplification	Audio Signal Analysis and Control
5	Motor Control, Feedback	Motor Control, Amplification
6	Audio Amplification	Light-Tracking Servo Control
7	Soldering and Circuit Assembly	Light-Tracking Servo

Table 2. Comparison of Lab Topics in 2013 and 2014 MITES curricula

Lab	E2@MIT
1	Linux Python I, II, Digital Inputs/Outputs
2	Analog Inputs/Outputs, Amplification
3	Audio Amplification, signal analysis, and Feedback Control

Table 3. E2@MIT Lab Topics

Final Projects (MITES and E2@MIT)

The second half of the MITES and E2@MIT courses were focused on designing, building, and debugging a final project that incorporated concepts learned in the course. A final presentation and demonstration of the projects in front of the entire MITES and E2@MIT communities was carried out on the final day. Final projects varied slightly between the MITES and E2@MIT EECS courses. In MITES, students are instructed to create a final project of their choosing, which can take the form of any device that interests them (determined in discussions with instructor to ensure feasibility). In E2@MIT, groups collectively self-assign themselves to one of a number of bio-medically themed projects, which they are then free to expand and customize. This is done because of time constraints; in MITES students spend approximately a week out of class brainstorming and researching ideas they would like to pursue, while in E2@MIT this isn't possible. Additionally, open-ended projects usually require the instructional staff to acquire outside materials after a decision on a project has been reached. This isn't possible in the one-week timespan of the E2@MIT curriculum. In both cases, however students are given significant freedom in what they actually deliver even if the project goal has been assigned. A selection of projects created from the past two years is shown in Figure 4 below with a brief overview of these are provided below. The type of single board computer used (BeagleBone Black or Raspberry Pi is specified and discussed in detail below) along with the year and specific program are listed.

MITES 2014
Lab 2: Inputs
June 18, 2014

Inputs and outputs in Embedded Systems:

The project for today is going to build upon what you learned last time.

Goal: *Build an electronic music lock box. You'll have a hidden code that you need to enter into the input switches (shown in Figure 1 below). If the code is correct, the system will play an mp3 file through some headphones. In the next lab, we'll build an audio amplifier for this system.*

On the Desktop will be a skeleton code file called mites_lab_2.py. Copy that into your Desktop folder that you used last time.

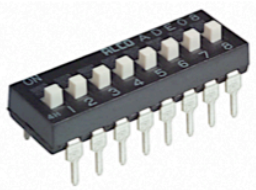


Figure 1. DIP Switch. When a switch is "ON" the pins across from eachother are connected.

GPIO Setup:
Remember from Monday how to set up the pins of the Raspberry Pi so that they are either inputs or outputs? Maybe not since we sort of avoided it.

- On the Gertboard, you connect Raspberry Pi "channels" (GP25, GP24, etc...) to the various "B" circuits. B1 has an outside-world interface of Buf1. B2 uses Buf2, etc...
- If you want B1 to serve as an output, put a jumper across the two B1 output pins Buf1 is now an output
- If you want B1 to serve as an input, put a jumper across the two B1 input pins. Buf1 is now an input.
- Doing both won't hurt anything, but will stop it from working
- So if we want GP25 to be an output and GP21 to be an input, we could hook up:
  - GP25 to B1, B1-output pins together, Buf1 is the output.
  - GP21 to B2, B2-input pins together, Buf2 is an input.

MITES 2014
Lab 2: Inputs
June 18, 2014

- You can mix and match GP pins and Buf circuits as needed. BX is always attached with BufX, however!!!!

Set up pins 25, 24,23,22, and 21 as inputs, both on the hardware and in the software. If you don't remember for the software, here's a helper (also look at your code from lab 1 if you want):

`GPIO.setup(number, direction, extra)`
`number` is any valid GPIO pin on the Raspberry Pi `direction` is either '`GPIO.IN`' or '`GPIO.OUT`' `extra` is only used for GPIO.IN, and we enter `pull_up_down=GPIO.PUD_UP`

You're going to need to read those inputs as you roll through this code. Remember the method to do this is:

`GPIO.input(pin)`
`Pin` is any valid GPIO pin Returns value measured at that pin (either 1 or 0 or True of False) `x = GPIO.input(24)` will assign the value at pin 24 to the variable x

Playing Music:
We'll be using the pygame package to play music files from our computers. In order to use it you'll need to type:

```
import pygame
```

at the top of the file up near the other import statements. Python by default only comes with certain packages pre-installed, but there are thousands of others that are already in existence that we can import and use for our own purposes. We'll be using quite a few libraries throughout the course.

In order to setup a pygame music player object in python you call the command:

```
pygame.mixer.init()
```

then you call on a music file that you have (here I have a semi-legally downloaded Kendrick Lamar mp3 that it is calling):

```
pygame.mixer.music.load('kendrick.mp3')
```

Then in order to play you call:

```
pygame.mixer.music.play()
```


MITES 2014
Lab 2: Inputs
June 18, 2014

The music will play for as long as the code runs or until a new call is placed on the `pygame.mixer` object.

Try playing some music:

Before you go about setting up some complicated stuff, go online to a site where you can readily get an mp3 file. You can use the browser on the Raspberry Pi...(DO NOT USE MIDORI...it is slow and doesn't download stuff). Save the file in a location where you will remember it. Ask for help.

Set up some simple code in the code skeleton that initiates, loads, and then plays the music. What do you notice happening?

Chances are no music played. That's because your code finished right after you set play so it never had a chance to run.

Give the program some time to run by calling `time.sleep()`

`time.sleep(x)`
`x` : the time in seconds that python will wait before continuing `time.sleep(0.5)` will delay by 500 milliseconds before continuing

Have your music file play for 20 seconds.

Today's Project:

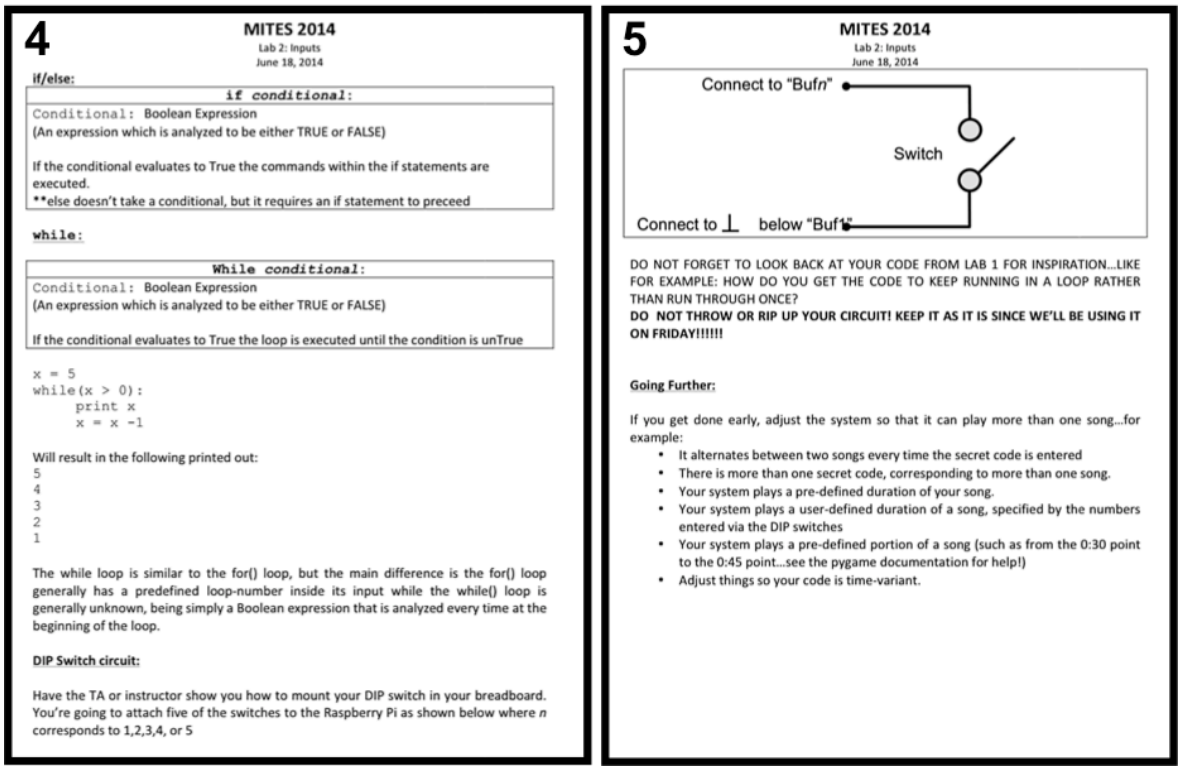
Today's project is going to now use what you just did with the music, and combine it with what you did on Monday (except with more switches).

- You will use five of the switches in the DIP switch package
- Each switch can either be ON or OFF
- When the switch is ON it makes the input to the Raspberry Pi go to 0
- When the switch is OFF, it makes the input to the Raspberry Pi go to 1
- Five switches means there are 2^5 =32 possible combinations.
- Choose one combination to be your secret code

Write your program so that it checks to see if the entered code is correct. If it is, start to play your music file until the code is turned off.

Some things that will help you in this project:

Figure 2. Extracted pages from Lab 2 in the Summer 2014 MITES EECS Course (continued in Figure 3).



4

MITES 2014

Lab 2: Inputs
June 18, 2014

if/else:

`if conditional:`
`Conditional:` Boolean Expression (An expression which is analyzed to be either TRUE or FALSE) If the conditional evaluates to True the commands within the if statements are executed. **else doesn't take a conditional, but it requires an if statement to preceed

`while:`

`While conditional:`
`Conditional:` Boolean Expression (An expression which is analyzed to be either TRUE or FALSE) If the conditional evaluates to True the loop is executed until the condition is unTrue

```
x = 5
while(x > 0):
     print x
     x = x -1
```

Will result in the following printed out:

```
5
4
3
2
1
```

The while loop is similar to the for() loop, but the main difference is the for() loop generally has a predefined loop-number inside its input while the while() loop is generally unknown, being simply a Boolean expression that is analyzed every time at the beginning of the loop.

DIP Switch circuit:

Have the TA or instructor show you how to mount your DIP switch in your breadboard. You're going to attach five of the switches to the Raspberry Pi as shown below where *n* corresponds to 1,2,3,4, or 5

5

MITES 2014

Lab 2: Inputs
June 18, 2014

DO NOT FORGET TO LOOK BACK AT YOUR CODE FROM LAB 1 FOR INSPIRATION…LIKE FOR EXAMPLE: HOW DO YOU GET THE CODE TO KEEP RUNNING IN A LOOP RATHER THAN RUN THROUGH ONCE?

DO NOT THROW OR RIP UP YOUR CIRCUIT! KEEP IT AS IT IS SINCE WE'LL BE USING IT ON FRIDAY!!!!!!

Going Further:

If you get done early, adjust the system so that it can play more than one song…for example:

- It alternates between two songs every time the secret code is entered
- There is more than one secret code, corresponding to more than one song.
- Your system plays a pre-defined duration of your song.
- Your system plays a user-defined duration of a song, specified by the numbers entered via the DIP switches
- Your system plays a pre-defined portion of a song (such as from the 0:30 point to the 0:45 point…see the pygame documentation for help!)
- Adjust things so your code is time-variant.

Figure 3. Extracted pages from Lab 2 in the Summer 2014 MITES EECS Course (continued from Figure 2).

- Response Time Game (BeagleBone Black, 2013 MITES, Fig. 4A): A game in which the objective is to "shoot" at a series of illuminated points using a laser pointer. LEDs serve as indicators and photoresistors serve as exposure detectors. The user has to shoot an indicated target within a certain period of time. Failure to do so results in loss of points. Successive rounds increase in speed and response time.
- Electrocardiogram (Raspberry Pi, 2014 E2@MIT, Fig. 4B): A functioning ECG built by the students and based off of an instructor-provided schematic. They then experimented and created a program that tracked, filtered, and analyzed the generated signal in order to estimate the patient's heart rate. A programmed threshold detector also generated an audio indicator of heart rate.
- Solar Charger (Raspberry Pi, 2014 E2@MIT, Fig. 4C): A solar-celled driven capacitor charger as a proof-of-concept of a larger solar-powered battery charger that could power a remote clinic. Students used the Raspberry Pi to track and estimate power usage, charge time, and send messages via Twitter to report the status of system.
- Robot with Object Avoidance (Raspberry Pi, 2014 MITES, Fig. 4D): A hybrid project featuring an Arduino controlling an ultrasonic range finder and a Raspberry Pi controlling a dual-motor robotic chassis. The robot was programmed

to go forward towards a light source until encountering an object in its path (determined through the ultrasonics) at which point it would execute an avoidance routine specified by the students.

- Artificial Insect Eye (BeagleBone Black, 2013 MITES, Fig. 4E): As part of a research project, students created a 64-element photo-sensing region using visible light phototransistors. A program was written to control hardware to scan the signals from all phototransistors and subsequently reconstruct a monochromatic image in a text-based display. Frame rate was approximately 10 Hz.
- Model Water Monitored Water Supply System (Raspberry Pi, 2014 E2@MIT, Fig. 4F): A functioning water supply system that would pump water up to an elevated holding tank from a simulated underground aquifer. Water level was monitored and tracked using sensors of the student design and water level and pressure were maintained via programming in the Raspberry Pi.
- Dance Dance Revolution Clone (Raspberry Pi, 2014 MITES, Fig. 4G): A dance pad, consisting of four pressure-sensitive switches. Students used elements of the Pygame package as a starting point for developing a dance-move response game complete with scoring and varying levels of difficulty.

Figure 4. Example Projects using BeagleBone Black and Raspberry Pi.

All student projects involved significant hardware and software engineering components. A brief overview of the program complexity developed by a number of groups is included below in Table 4.

Group	Num. of Lines of All Code*	if/else control structure?	for, while loops?	Function abstraction?	Class abstraction
1	42	Yes	Yes	No	No
2	34	Yes	Yes	Yes	No
3	253	Yes	Yes	Yes	Yes
4	21	Yes	Yes	No	No
5	88	Yes	Yes	Yes	No
6	51	Yes	Yes	No	No
7	33	Yes	Yes	No	No
8	55	Yes	No	No	No
9	60	Yes	Yes	Yes	No
10	152	Yes	Yes	Yes	Yes
11	76	Yes	Yes	No	No
12	90	Yes	Yes	Yes	No
13	38	Yes	Yes	No	No

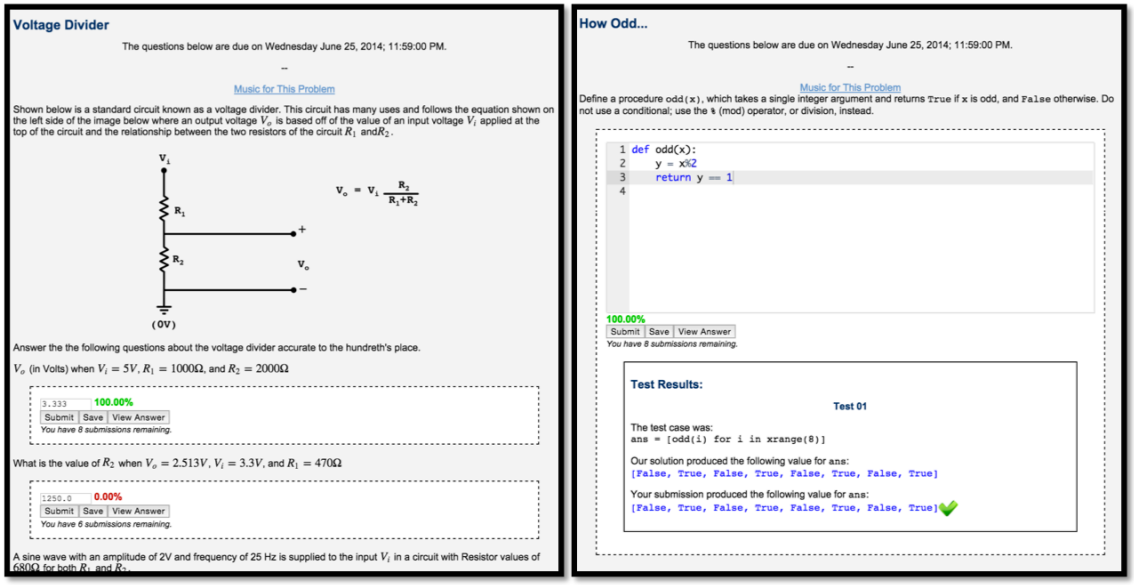
Table 4. A simple analysis of student programs written for their final projects.
*Estimated by ignoring large gaps in code.

Python Programming Language

Part of the motivation to move to single-board computers was to allow the use of Python as the primary programming language[15]. We adapted an online teaching environment that had previously been developed by Adam Hartz at MIT for one of its large EECS introductory courses known as CAT-SOOP (CAT-SOOP is an Automated Tutor for Six-Oh-One Problems)[20]. This learning environment provided a platform in which students explored hardware and software concepts, answered questions, and tested code out in preparation for labs on the following day (Figure 5A and 5B). While both the Raspberry Pi and the BeagleBone Black come with Python preinstalled, in order to access and interface with the two system's hardware, we needed to create wrapper libraries simple enough for students to use.

BeagleBone Black vs. Raspberry Pi

While we felt use of single-board computer-based platforms for lab exercises and PBL was a success in 2013, we ran into several serious issues with the BeagleBone Black. The BeagleBone Black was difficult for students to work with because it lacked clear labeling of its connections. We resorted to attaching colored tape and providing hookup diagrams customized for each lab exercise in order to streamline student experience as shown in Figure 6. Second, the lack of readily accessible audio output from the board prevented students from integrating audio or music into their projects (audio output only being accessible via HDMI, and we could not generate a functional breakout of this due to time constraints). Third, and by far the most frustrating, was the sensitivity of the BeagleBone



(A) **(B)**

Figure 5. An web-browser accessible Python and engineering automated tutor, CAT-SOOP, was adapted to provide lessons for students in (A) hardware and (B) software-focused problems.

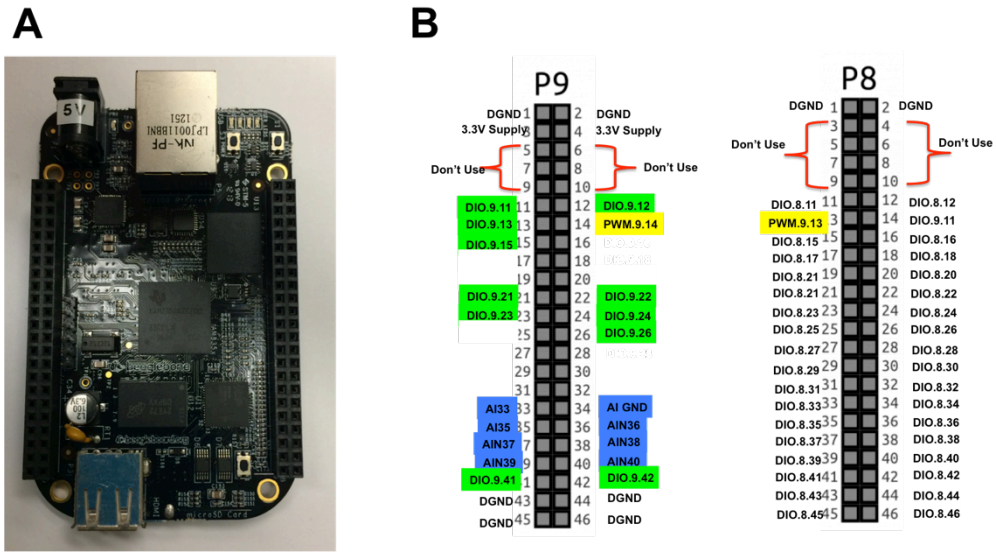


Figure 6. (A) BeagleBone Black Revision B, which were used for 2013 summer programs (B) Example Pin diagram provided to students to assist in interfacing with BeagleBone Black.

Black to external voltages. Because many labs and final projects interfaced the computer directly to breadboarded circuits, ranging from simple devices such as LEDs, to complex ones including coin-operated switches, we inadvertently created a laboratory environment with board-incompatible voltages (greater than 3.3V). While not dangerous for the students, this did lead to undesirable behavior of the BeagleBone Black computers, ranging from sudden shutdown of the machine to permanent destruction of the entire

board. In the summer of 2013, throughout all courses, we lost five out of ten BeagleBone Black computers in such a way. While similar failure modes had been experienced previously when we working with the Arduino platform, it was relatively easy to fix since student programming work was saved on external computers, and the ATMega328 DIPs (preloaded with an Arduino boot loader) could be readily replaced on the Arduino Duemilanove and Uno models that we used at a cost of approximately 5 USD. In the case of the BeagleBone Black single-board computers, the surface mount nature of the chips prevented repairs. Furthermore, all work by the students was lost and no longer accessible due to the integrated onboard flash memory. This proved extremely frustrating for several groups, particularly in the MITES program, where student Python programs often were above 100 lines of code.

Improvements for 2014

For 2014, we adjusted pieces of the infrastructure to address issues from the prior year's implementation. Priority was centered on input/output protection of the computers. The only suitable such board we could find was the Gertboard, intended for use with the Raspberry Pi and designed by Gert Van Loo. The Gertboard is available at many hobby electronics retailers that supply Raspberry Pis and assorted peripherals and can be purchased for approximately 50 USD as of 2014[18,19]. It can mount directly onto the Raspberry Pi, however we chose to connect them using a 26-pin ribbon cable as shown in Figure 7. This had the added benefit of providing some distance from the Raspberry Pi and student electronics. In addition to providing input and output protection for the GPIO pins, the Gertboard also came with an on-board digital-to-analog converter (DAC), a dual-channel analog (ADC), an onboard ATMega328 (Arduino-compatible microprocessor), and a single-channel motor driver. All of these devices could be utilized through the selection of a series of jumpers placed on the board as well as a software library provided by the developers. While adopting the Gertboard required switching from the BeagleBone Black to the Raspberry Pi, the benefit of input/output protection and additional features of the Gertboard were enough to justify rewriting our 2014 curricula to utilize the Raspberry Pi and Gertboard in spite of the significant additional work it required.

In redesigning our labs for use with the Raspberry Pi/Gertboard, an immediate challenge was how to deal with the complex jumper/hookup diagrams of the Gertboard necessary for different functionalities. For example, to use a specific Input/Output (I/O) pin as a digital output required several lines of setup code as well as the placement of a two wire jumpers. This effectively protected the Raspberry Pi I/O from shorts and overvoltage exposure, but proved far too difficult for students, especially at the beginning of a course or in a time-constrained workshop, to implement. The solution to this problem was to pre-wire Gertboards for each lab's exercises and provide a pre-made software library that would be imported in each day's program skeleton provided to the students. For the software distribution, all machines were either manually synchronized or linked to a Git repository which allowed rapid distribution (either manually or by default at startup) of current code distributions for the day's assignments. For hardware setup, organization of the jumper configuration was generally carried out by the course TA and instructor

immediately before class began, taking approximately one minute per board. In the case of MITES and E2@MIT, over the course of the labs, we spent small amounts of time describing actual details of the Gertboard, and only focused on their specific workings during the final project periods as required by particular student projects.

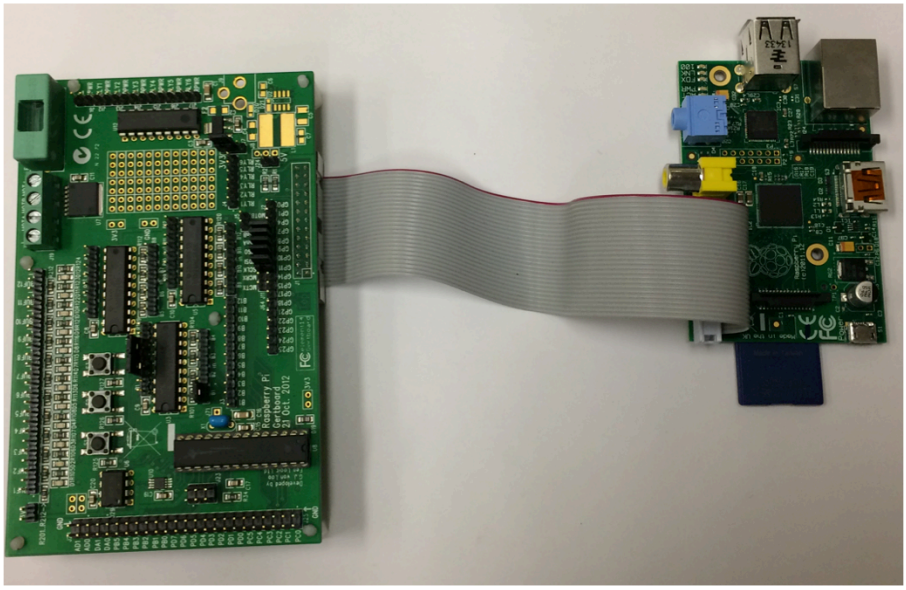


Figure 7. Raspberry Pi Model B (right) connected to a Gertboard via ribbon cable (left).

Deployment of the Raspberry Pi/Gertboard assemblies in 2014 was as successful than 2013's BeagleBone Blacks in terms of student progress and project complexity. In addition, the increased usability and protection afforded by the Gertboard completely removed the danger of partial/complete loss of student work, and greatly improved the speed and efficiency of student wiring when interfacing with the board due to the clear hookup pin numbering.

	Approximate Cost (USD)	
Item	**BeagleBone Black**	**Raspberry Pi**
Main Board	50	35
External Shield	N/A	50
Keyboard/Mouse	15	15
USB Extension	5	5
SD card	N/A	6
Power supply	9	9
Monitor connector/adaptor	12	12
Monitor	N/A*	N/A*
Total:	91	132

Table 5. Total cost of components for single-board setup of BeagleBone Black and Raspberry Pi. *Monitors were borrowed or acquired for free for our course. Sufficient models of monitor for both platforms can be purchased for approximately 100 USD.

Setup Costs

In comparing the two systems, as implemented between the two years, the BeagleBone Black was the less expensive (91 USD vs. 132 USD) as shown in Table 5, however because we lost 50% of the BeagleBone Blacks due to in-lab accidents the cost benefit narrows significantly. Furthermore, a simplified output protection board will be used in 2015, further decreasing the cost of a Raspberry Pi to at or below the cost of the BeagleBone Black setup. The Raspberry Pi/Gertboard assembly was far more durable in-lab, however this isn't a completely fair comparison since the Raspberry Pi on its own is just as sensitive to overvoltage exposure and accidental shorts as the BeagleBone Black boards (assessed through in-lab stress tests by the author), and one could argue that with an equivalent, compatible Gertboard-equivalent for the BeagleBone Black, the shorting problems would have been avoided.

In our experience, audio-based experiments and final projects have routinely been among the most popular final projects and activities carried out by students, and the much more accessible audio interface of the Raspberry Pi (3.5 mm TRS connector) proved to be a significant benefit than when working with the BeagleBone Black. Additionally, numerous Python packages have been developed and tested specifically for use on the Raspberry Pi, and this made assisting student work far easier in the time-restricted environment of these summer courses. The added complexity of the Gertboard hardware and software did prove to be a hurdle for many students during the course, however in our opinion its benefits outweigh these additional challenges. Taking everything into consideration, we intend to continue using the Raspberry Pi/Gertboard platform for future iterations of this course

MOSTEC Workshops

Our third and shortest single-board curriculum from recent years is an EECS-themed workshop run for the residential portion of the MOSTEC program using the hardware platforms derived for the other two courses. These workshops were three hours long and involved a brief discussion of the EECS field, a cleanroom tour at MIT (given by Jamie Teherani), and about two hours of in-lab explorations. Because of the tight time constraints labs are significantly more guided than in E2@MIT or MITES, however we did leave them open-ended by proposing a number of improvements at the end of each activity. The amount of guidance provided in the MOSTEC laboratory handouts was greater than in either the E2@MIT or MITES exercises. This allowed students with a range of different backgrounds the opportunity to achieve some subset of lab goals. Somewhat similar labs have been used in both years, with BeagleBone Blacks used for 2013 and Raspberry Pi systems used in 2014. In both situations, eight stations were set up and students were free to work in groups of their choosing of two or three. The three primary projects we've had in the workshops have been:

- Temperature Logger: Students must interface with an LM35 temperature sensor and then write code to sample interpret output of sensor and then email updates to a personal address. Students can expand and customize the project through use of

multiple temperature sensors, switches, etc. An example of the starter code skeleton and one student group's solution in 2014 is shown in Figure 8A and 8B, respectively.

- Reaction Timer (2014 only): Students are tasked with creating a sound and light response-time circuit that measures how quickly a human can respond to a light or sound stimulus (flash of LED or beep from buzzer, respectively).
- Audio Amplifier: Students are tasked with building an LM386 audio amplifier using discrete components on a breadboard, and then encouraged to interface it with Pygame-based audio playing code.

```
from MOSTEC import * #import important MOSTEC
files

sendToAddress = 'default@mit.edu' #who you want to
email

while True: #run forever
    value = readAnalogInput() #read in value at
Analog Input 1
    print 'hello' #print 'hello'
    #your code here (consider getting rid of
pointless "hello" eventually)
```

(A)

```
from MOSTEC import * #import important MOSTEC
files

sendToAddress = 'default@mit.edu' #who you want to
email

while True:
    value = readAnalogInput()
    value = value*3300.0/1023
    tem = value/10
    print tem
    delay(8000)
    if temp>29:
        writeDigitalOutput(1,1)
        message = 'Current Temp is'+str(temp)+'
deg C.'
        Server=smtplib.SMTP('smtp.gmail.com:587')
        server.starttls()
        server.login(username,password)

server.sendmail(fromAddress,sendToAddress,message)
        server.quit()
    else:
        writeDigitalOutput(1,0)
```

(B)

Figure 8. (A) Example of temperature-logging skeleton code provided to students in 2014 MOSTEC Electronics workshop, and (B) resulting final working code developed by a group of three students.

Conclusions, Scalability, Applicability, and Future Work

Overall we are pleased with the last two years of curriculum development. Aside from some minor changes, we believe the hardware implementation for the 2015 versions of the course will remain relatively stable compared to what is reported here. The complexity, and variety of projects that students have been capable of generating using single-board computers is much greater than when compared with prior years. This is readily apparent when comparing the complexity of projects in this paper with the versions of the E2@MIT course that used Arduinos shown elsewhere[5]. Student feedback from both the 2014 MITES class and the 2014 E2@MIT class was anonymously collected at the completion of both courses. The results were generally positive (Table 6, Table 7, and open-ended student feedback in Table 8). We currently (as of the time of writing this draft) do not have access to student feedback from 2013 and from both years of the MOSTEC workshop.

We have two major goals for the upcoming summer of 2015 that will expand upon the curriculum development discussed in this paper. The first is to collect large amounts of

quantitative student performance data from before, during, and after the courses in order to more effectively assess student progress during the course and to more effectively asses the efficacy of the curriculum as a whole. Previously, content-specific assessment of student activity was not possible in the scope of OEOP enrichment programs, but will be possible for this upcoming year. Additionally because students perform all work using the single-board computer-based development platforms (Figure 1) that we've developed, we can record student activity in real time in the laboratory setting. This development work is ongoing and will be accomplished by integrating the Raspberry Pi hardware with the previously developed tutor system (CAT-SOOP) referenced above in this paper. Such a setup will permit the collection of real-time data of student lab progress from both the software and hardware side of their activities. The ultimate plan is to provide both hardware and software "test-cases" that will help students get more feedback as progress through the guided portions of the labs, and provide the instructional staff with a means of monitoring additional work the students perform in class.

A second major goal for this upcoming summer is to implement a completely remote version of the single-board computer curriculum through an additional program at the OEOP. We began to develop something somewhat similar in the summer of 2014, creating an online EECS curriculum that focuses on both hardware and software using a previously developed remote hardware infrastructure (iLab)[21,22], hardware simulations, and the CAT-SOOP remote automated tutor discussed above[3,20]. For the summer of 2015 we intend to send all participating students a hardware development kit similar to that shown in Figure 1, which they will then set up and run locally at their homes. All of the distributed systems will be connected and synchronized with a central server on MIT's campus. Rather than have students work with simulated circuits over their computer as was done in 2014, students will carry out hardware and software-based tasks locally on their own version of the development environment[23]. By merging this with the hardware and software monitoring we're developing for the residential programs we will be able to provide students feedback on their progress of labs in an automated fashion remotely. This work is on-going and we intend to have more details to show at the ASEE Conference in June or 2015.

Prompt	Strongly Agree	Agree	Undecided	Disagree	Strongly Disagree
"I learned a lot in this course"	100%	0%	0%	0%	0%
"I was challenged in this course"	91.7%	0%	8.3%	0%	0%
"I would recommend this course to others"	83.3%	16.7%	0%	0%	0%
"The instructor of this course helped me learn in a different way"	91.7%	8.3%	0%	0%	0%
"The instructor spent most of class time lecturing"	75.0%	16.7%	8.3%	0%	0%
"The course was appropriate for my ability level"	91.7%	8.3%	0%	0%	0%

Table 6. Post-program responses of students to MITES EECS curriculum (2014) (n=12).

Prompt	Strongly Agree	Agree	Undecided	Disagree	Strongly Disagree
"I learned a lot in this course"	91.7%	8.3%	0%	0%	0%
"I was challenged in this course"	83.3%	16.7%	0%	0%	0%
"I would recommend this course to others"	91.7%	8.3%	0%	0%	0%
"The instructor of this course helped me learn in a different way"	91.7%	8.3%	0%	0%	0%
"The instructor spent most of class time lecturing"	25.0%	75.0%	0%	0%	0%
"The course was appropriate for my ability level"	66.7%	25.0%	8.3%	0%	0%

Table 7. Post-program responses of students to E2@MIT EECS curriculum (2014) (*n*=14).

"The most exciting and enjoyable course of my MITES experience. Put real-life application to previous knowledge in science and math."
"On top of that, the class mostly consisted of us experimenting with wires and circuits. We learned through hands-on experiences, which greatly helped us in completing the final project which was so fun to work on itself. We behaved like a family, and I was able to know everyone in the class on a deeper level..."
"I still cannot believe that someone without any previous experience in Python and EE could actually help build a thermal cycler in a course of two days. I am amazed at how amazing the people were in the class, especially the TA and our instructor. I loved the experience and would recommend it to my friends…"
"I really, really enjoyed the electronics course!...I learned a lot and had a lot of fun. Before this experience, I was set on going into medicine, but now I am considering electrical engineering, as well."
"I really enjoyed being in the Electronics course…It didn't feel like an ordinary course where the only objective is to learn. I was also able to receive as much help as I needed, so it did not matter that I started the course with little previous knowledge of the material."
"Electronics was not my first choice coming into E2. However, I now regret that is was not because if I had not been placed in electronics, I would have missed out on an amazing experience. With little prior knowledge of programming, the tasks were difficult at first. But by the end of the week, I managed to help build and ECG that functioned extremely well. Never, when I was told about the course, did I think we were going to be able to finish the projects in two days. The fact that we were was due to the staff and students in the class. No student was off task…"

Table 8. Open-ended feedback from 2014 MITES and E2@MIT EECS participants.

Acknowledgement

This work would not have been possible without assistance from the OEOP staff including Shawna Young, Sandra Tenorio, Professor Cardinal Warde, Yonatan Tekleab, Nathnael Abebe, Stephanie Haro, as well as Adam Hartz, and MIT's EECS Department.

Bibliography

1. Thomas, J. W. *A Review of Research on Project-Based Learning*. (2000).
2. Laboy-Rush, D. *Integrated STEM Education through Project-Based Learning*. at <http://www.rondout.k12.ny.us/common/pages/DisplayFile.aspx?itemId=16466975>

3. Steinmeyer, J. D. Online EECS Curriculum for High School Students. In *Integrated STEM Education Conference (ISEC), 2015 IEEE 5th* (2015).
4. Brock, J. D., Bruce, R. F. & Reiser, S. L. Using Arduino for introductory programming courses. *J. Comput. Sci. Coll.* **25,** 129–130 (2009).
5. Steinmeyer, J. D. Accelerated project-based introduction to EECS for high school students. In *Integrated STEM Education Conference (ISEC), 2012 IEEE 2nd* (2012).
6. Brock, J. D., Bruce, R. F. & Cameron, M. E. Changing the world with a Raspberry Pi. *J. Comput. Sci. Coll.* **29,** 151–153 (2013).
7. Rosen, W., Ertekin, T. & Carr, M. E. An Autonomous Arduino-Based Racecar for First-Year Engineering Technology Students. in *American Society of Engineering Education Annual Conference* (2014).
8. Walter, W. W. & Southerton, T. G. Teaching Robotics by Building Autonomous Mobile Robots Using the Arduino. in *American Society of Engineering Education Annual Conference* (2014).
9. Kim, M. & Schlosser, J. Hardware Support for Project Based Learning. In *IEEE EDUCON Pre-Conference Workshop* (2013).
10. Lau, K. W., Erwin, B. T. & Petrovic, P. Creative learning in school with LEGO(R) programmable robotics products. in *FIE'99 Frontiers in Education. 29th Annual Frontiers in Education Conference. Designing the Future of Science and Engineering Education. Conference Proceedings (IEEE Cat. No.99CH37011* **2,** 12D4/26–12D4/31 (Stripes Publishing L.L.C, 1999).
11. Erwin, B., Cyr, M. & Rogers, C. LEGO Engineer and RoboLab: Teaching Engineering with LabVIEW from Kindergarten to Graduate School. at <http://www.researchgate.net/publication/246676439_LEGO_Engineer_and_RoboLab_Teaching_Engineering_with_LabVIEW_from_Kindergarten_to_Graduate_School>
12. Williams, A. B. The qualitative impact of using LEGO MINDSTORMS robots to teach computer engineering. *IEEE Trans. Educ.* **46,** 206–206 (2003).
13. Recktenwald, G. W. & Hall, D. E. Using Arduino as a Platform for Programming, Design, and Measurement in a Freshman Engineering Course. in *American Society of Engineering Education Annual Conference* (2011).
14. Bird, N. Use of the Arduino Platform for a Junior-Level Undergraduate Microprocessors Course. in *American Society of Engineering Education Annual Conference* (2011).
15. Zelle, J. M. *Python as a First Language*. at <http://mcsp.wartburg.edu/zelle/python/python-first.html>
16. Ateeq, M., Habib, H., Umer, A. & Rehman, M. U. C++ or Python? Which One to Begin with: A Learner's Perspective. in *2014 International Conference on Teaching and Learning in Computing and Engineering* 64–69 (IEEE, 2014). doi:10.1109/LaTiCE.2014.20
17. Wirth, M. & McCuaig, J. Making Programs With The Raspberry Pi. in *Proceedings of the Western Canadian Conference on Computing Education - WCCCE '14* 1–5 (ACM Press, 2014). doi:10.1145/2597959.2597970
18. Foundation, R. P. Gertboard.
19. Van Loo, G. & VanInwegen, M. Gertboard User Manual Revision 2.0. (2012).
20. Hartz, A. CAT-SOOP : a tool for automatic collection and assessment of homework exercises. (2012). at <http://dspace.mit.edu/handle/1721.1/77086>
21. Mitchell, R., Fischer, J. & del Alamo, J. A. A Survey Study of the Impact of a MIT Microelectronics Online Laboratory on Student Learning. in *ASEE/IEEE Frontiers in Education Conference* (2006).
22. Fischer, J., Mitchell, R. & del Alamo, J. Inquiry-Learning with WebLab: Undergraduate Attitudes and Experiences. *J. Sci. Eng. Technol.* **16,** 337–348 (2007).
23. Circuits, 123D. *Simulate Arduino online and easily create custom circuit boards*. (2014).